OCR
Oxford Cambridge and RSA

An OCR endorsed teaching and learning tool

OCR A Level Computer Science

H446 – Paper 2

4

Subroutines

Unit 11

Programming techniques

PG ONLINE

Objectives

- Be familiar with subroutines (functions and procedures), their uses and advantages
- Use subroutines that return values to the calling routine
- Use parameters to pass data to subroutines by value and by reference
- Contrast the use of local and global variables

Subroutine

- A subroutine is a set of instructions with a name
- What will the following code produce?

```
procedure goodBye()
   print("It has been nice meeting
you.")
endprocedure

procedure hello()
   print("Hello, how are you today?")
endprocedure

# main program starts here
hello()
print("I am a computer program.")
goodBye()
```

Subroutine

- Program execution starts at the first statement in the main program
- Program flow will “jump” to the subroutine when called
- When the subroutine has finished, the program will continue from where it was called
- Procedures and functions are types of subroutine
  - A function is a subroutine that returns one or more values
  - In some programming languages all subroutines are

Built-in functions

- Programming languages come with many built-in functions which perform common tasks
- Here are some examples - can you guess what they do?

```
x = sqrt(num)
print("Print this message to the screen")
y = len("Bob")
c = asc("a")
b = chr(65)
Z = round(6.5743, 2)
```

Library subroutines

- Programming languages also come complete with libraries of pre-defined subroutines
- The module library has to be imported at the start of the program
- For example, in Python:

```
import random
x = random.randint(1,6)
```

  - What will this put in x?
  - What is the name of the function?

Passing arguments

- Arguments in parentheses are used to pass data to a subroutine
  - What values are passed to the subroutine in the example below?
  - What identifiers are used to hold the values?

```
procedure multiply(x,y)
    print(“The product is: ”, x * y)
endprocedure

#call subroutine
multiply(2,5)
```

Parameters and arguments

- The terms parameter and argument are often used interchangeably
- Strictly speaking, parameters appear in subroutine definitions, and arguments appear in subroutine calls
- Look back at the code on the previous slide
- Can you identify the parameters and arguments?
- The arguments may vary from call to call, but the parameters are part of the subroutine definition

Subroutine parameters

- The order of the parameters in the parentheses in the subroutine is important
  - What is output here?

```
procedure calc (x, y, z)
    ans = (x * y) + z
    print(ans)
endprocedure

#call subroutine
calc(2,5,6)
calc(2,6,5)
```

Passing arguments

- What is the output from this program?

```
procedure proc1 (x, y)
   x = x – 2
   y = 0
endprocedure
#main program
m = 8
n = 10
proc1(m, n)
print (m, n)
```

Passing by value

- In some programming languages (such as Python), all arguments are passed by value
- This means that the actual value of the argument in the calling statement is copied to the variable parameter in the subroutine
- Any calculation performed on that parameter in the subroutine will not affect the value of the corresponding argument in the calling routine

Passing by reference

- Some programming languages allow arguments to be passed by value or by reference
- “By reference” means that the address of the argument in the calling statement is passed to the corresponding parameter in the subroutine
- Any calculation performed on that parameter in the subroutine will change the value of the corresponding argument in the calling routine

Example

- What is the output from this program?

```
procedure proc1 (By val:x, By ref:y)
    x = x – 2
    y = 0
endprocedure
#main program
m = 8
n = 10
proc1(m, n)
print (m, n)
```

Worksheet 4

- Complete **Task 1**

Functions returning a value

- A function can return one or more values using a return statement
  - Compare the code samples below. How are they different?

```
procedure multiply(x,y)
    print("Product:")
    print(x*y)
endprocedure

#main program
multiply(2,5)
```

```
function multiply(x,y)
    product = x*y
    return product
endfunction

#main program
answer = multiply(2,5)
print("Product:, answer")
```

Functions returning a value

- Try the code below with the two sets of values:
  - a = 2, b = 2, c = 2 a = 3, b = 5, c = 4

```
function sum(x,y,z)
    calc = x*y + z
    return calc
endfunction

if sum(a,b,c) > 10 then
    print("larger than 10")
else
    print("less than 11")
endif
```

Variable scope

- When a variable is in scope the values can be accessed
  - What might happen in this code snippet at the various lines?
  - Which line will give an error?
  - What is a local variable?
  - What is a global variable?

```
1 function swap(a,b)
2     temp = a
3     a = b
4     b = temp
5     return a,b
5   endfunction

6   x = 2
7   y = 3
8   print(x,y)
9   x,y = swap(x,y)
10  print(x,y)
11  print(temp)
```

Local and global variables

- A subroutine may have its own variables, like temp in the subroutine shown on the previous slide
- These are known as local variables
- A global variable is defined in the main program and can be used in any subroutine called from the main program
- When you use a name on the left-hand side of an assignment statement in a subroutine, a local variable is automatically created
- A local variable can be used only in that subroutine

Scope of variables

- The scope of a local variable is the subroutine in which it is declared
- The variable does not exist outside the subroutine
  - What happens here?

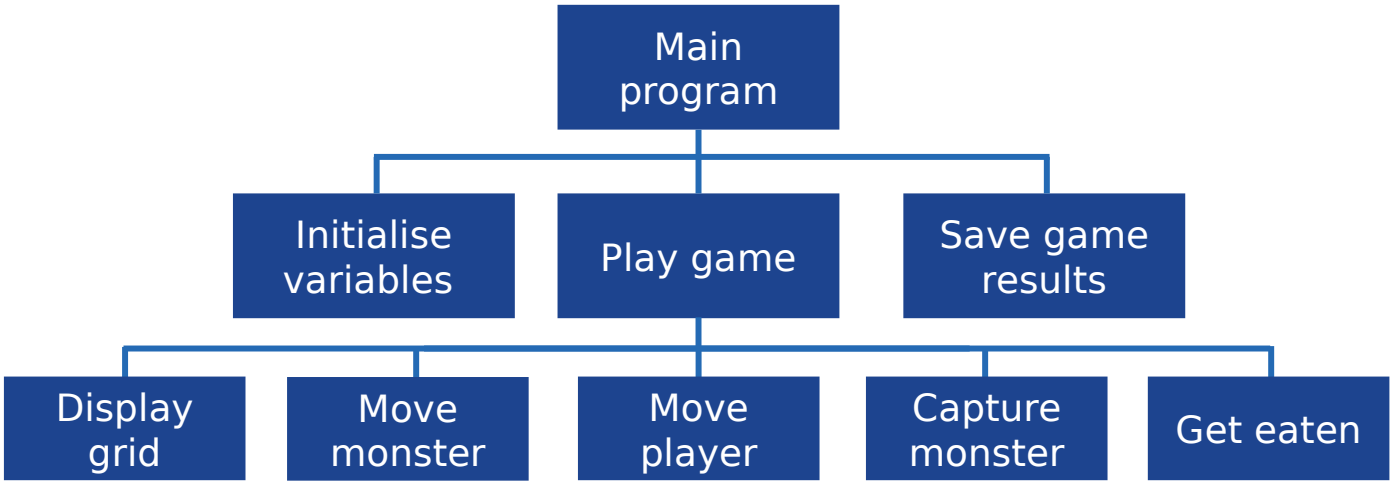
```
procedure printnumber
    num = 15
    print(“num = ”,num)
endprocedure

// main program
printnumber
output (“num = ”, num)
```

Advantages of local variables

- The subroutines will be independent of a particular program and can be re-used in different programs
- There is no chance of accidentally changing a variable in the main program that is used in a subroutine or vice versa
  - Keep your subroutines self-contained!
  - Pass as arguments any values that are needed

Modular programming

- Modular programming means breaking down a major task into smaller subtasks
- These subtasks may be further broken down until each ‘module’ performs a single function



Advantages of modular programming

- Programs are more easily and quickly written
  - Large programs are broken down into subtasks that are easier to program and manage
  - Each module can be individually tested
  - Modules can be re-used several times in a program
  - Large programs are much easier to debug and maintain
- Can you think of any other advantages?

Worksheet 4

- Now try the questions in **Task 2**

Plenary

- Here’s what you should be able to do!
  - Use subroutines (functions and procedures), and describe their uses and advantages
  - Use subroutines that return values to the calling routine
  - Use parameters to pass data to subroutines by value and by reference
  - Contrast the use of local and global variables

Copyright

Licence agreement

PG ONLINE